Sections and Chapters

Gubert Farnsworth

Ngày 14 tháng 12 năm 2020

STRING MATCHING

1 String Matching Problem

Tìm kiếm chuỗi hiện diện trong cuộc sống, ví dụ như: tìm tên thầy dạy DSA trong danh sách giảng viên, tìm tên trong bảng điểm,.. hoặc trong khoa học, như là tìm liệu cấu trúc ADN của virus này có trong virus khác hay không,

2 String Matching Algorithms

1. Brute-force

   Giải thuật Brute-Force, hay còn gọi là vét cạn, là thuật toán đơn giản nhất trong các thuật toán tìm kiếm chuỗi con *pattern* trong chuỗi cha *text*.

   Có thể giải thích đơn giản, giải thuật Brute-Force so sánh lần lượt mỗi chuỗi con *subtext* của *text* có cùng chiều dài với *pattern* với *pattern*, nếu tìm được, trả về kết quả là vị trí được tìm thấy; khi không tìm được kết quả mong muốn, trả về giá trị quy ước là không tìm thấy.

   Trong ví dụ sau, ta sẽ làm rõ cách hoạt động của giải thuật này:
   text = `Let them go!`
   pattern = `them`

```
Let them go!
them
```

```
Let them go!
 them
```

```
Let them go!
  them
```

```
Let them go!
   them
```

```
Let them go!
    them
```

```
Let them go!
     them
```

```
Let them go!
     them
```

```
Let them go!
     them
```

```
Let them go!
     them
```

Ta tìm thấy chuỗi pattern tại vị trí thứ 4!

Từ ví dụ trên, ta thiết kế mã giả cho giải thuật Brute-Force:

```
        vị trí tìm thấy = -1
        subtext = chuỗi con đầu text có độ dài bằng pattern
        while (chưa tìm thấy hoặc chưa tới cuối text)
          if (từng ký tự của subtext = pattern):
            trả về vị trí
          else:
            dịch chuyển chuỗi con subtext trong text sang phải 1 chữ cái
        Trả về: vị trí tìm thấy
```

Với chuỗi *pattern* có độ dài là M, chuỗi *text* có độ dài là N
Phân tích độ phức tạp của thuật toán trong trường hợp xấu nhất:

- Mỗi lần so sánh với *subtext*, *pattern* phải so sánh nhiều nhất là M lần (trong trường hợp cả M - 1 ký tự đầu đều đúng).
- Có tất cả N - M + 1 chuỗi, vậy số chuỗi cần so sánh nhiều nhất là N - M + 1 *subtext* như vậy (trong trường hợp N - M + 2 chuỗi *subext* đầu không trùng với *pattern*)
  → Cần M(N - M + 1) lần. Vì duyệt tới cuối mảng nên đây là trường hợp tìm thấy ở cuối mảng, hoặc không tìm thấy
  → Cận trên $O(MN)$ (vì N > N - M + 1).
  → Cấp phát bộ nhớ: 0.

Phân tích độ phức tạp của thuật toán trong trường hợp tốt nhất:

- Trong trường hợp tốt nhất, có thể thấy *pattern* chính là *subtext* đầu tiên của *text*.
- Như vậy, chỉ cần tốn M lần so sánh các ký tự.
  → Cần M lần.
  → Cận trên $O(M)$.
  → Cấp phát bộ nhớ: 0.

Đánh giá:

- Dễ hiểu, thuật toán này chỉ duyệt từ đầu đến cuối, so sánh tuần tự từng chuỗi con với chuỗi cần tìm kiếm.
- Không cần bước tiền xử lý (như các thuật toán được trình bày bên dưới).
- Độ phức tạp $O(MN)$. Không cần xin thêm bộ nhớ.

2. Rabin-Karp

3. Knuth-Morris-Pratt

   Giải thuật Knuth-Morris-Pratt, về cơ bản cũng giống như thuật toán Brute-Force, tuy nhiên, chỉ khác ở chỗ, Brute-Forch khi so sánh *pattern* và *subtext*, Brute-Force so sánh toàn bộ các ký tự của chúng lại từ đầu, còn Knuth-Morris-Pratt, từ lần so sánh trước, sẽ quyết định có so sánh pattern với *subtext* kế tiếp không, và nếu không sẽ nhảy bao nhiêu bước đến *subtext* khác. (đã biết được chúng giống nhau), từ đó tiết kiệm được chi phí giải bài toán.

   Để dễ hiểu, ta xét 1 ví dụ như sau:

   text = abacababd
   pattern = abab

```
abacababd
abab                    // So sánh như Brute-Force
```

```
abacababd
abab
```

```
abacababd
abab
```

abacababd
abab // ta thấy ký tự thứ 4 không trùng rồi
//

Let␣them go!
them

Let them go!
them

Let them go!
them

Let them go!
them

Let them go!
them

PROGRAMING

(abc)

1 Introduce

2 Example Test